RDeco Package

Jack Carter, Samuel Davenport, Jeremias Knoblauch, Giulio Morina

DECO Algorithm

DECO Algorithm provides a way to compute Lasso regression coefficients in a parallel and distributed way when $p \gg n$, where p is the number of covariates and n is the number of observations. The algorithm is based on splitting the $n \times p$ matrix X vertically in m submatrices.

The theoretical fundation behind the algorithm can be found in "DECOrrelated feature space partitioning for distributed sparse regression" paper by Wang, Dunson, and Leng (2016).

RDeco package

`RDeco` package provides 4 different implementations of the DECO algorithm:

- `DECO_LASSO_R`: a pure R implementation;
- `DECO_LASSO_MIX`: a mix of R and C++ implementation;
- `DECO_LASSO_C_PARALLEL`: a pure C++ implementation;
- `DECO_LASSO_R_CLUSTER`: a pure R implementation that splits the work load on different machines (still not stable when X is big).

All four functions currently only accept a fixed penalty parameter λ and do not implement an automatic way to tune it.

Given the same dataset and same parameters, the four functions all return the same result:

```
#Generating a simulated dataset
set.seed(100)
n<-5
p<-17
m<-4
sigma<-1
lambda<-0.03
r<-0.01
X<-matrix(rnorm(n*p, sd=1), nrow=n)
eps<-rnorm(n, mean = 0, sd = sigma)
activeCoefs<-sample(c(0,1),p, prob=c(0.5, 0.1), replace=TRUE)
coefs<-activeCoefs*1
Y<-X%*%coefs + eps
ncores <- 4
clust <- makePSOCKcluster(c("greywagtail","greyheron","greypartridge","greyplover"))
for (refinement in c(FALSE,TRUE)) { #Check that the returned value is the same
  #when the refinement step is performed and when is not.
  res<-DECO_LASSO_R(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,r_2=0.001,
                    ncores = ncores, refinement = refinement)
  res_mix<-DECO_LASSO_MIX(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,r_2=0.001,
                          ncores = ncores, refinement = refinement)
```

```
  res_C<-as.vector(t(DECO_LASSO_C_PARALLEL(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,
                                           r_2=0.001, ncores = ncores, refinement = refinement)))
  res_cluster <- DECO_LASSO_R_CLUSTER(Y, X, p=p, n=n, lambda=lambda, r_1=r, r_2=0.001,
                                      clust=clust, ncores=ncores, refinement = refinement)
  if(all.equal(res,res_mix) == TRUE && all.equal(res,res_C) == TRUE
     && all.equal(res_C,res_cluster) == TRUE) {
    print("All functions return the same result!")
  } else {
    print("NOT all functions return the same result...")
  }
}
```

```
## [1] "All functions return the same result!"
## [1] "All functions return the same result!"
```

```
stopCluster(clust)
```

How to perform DECO algorithm: examples and options

DECO_LASSO_R

```
  DECO_LASSO_R(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,r_2=0.001, ncores = ncores,
               refinement = TRUE, intercept=TRUE)
```

Note that it is possible to include the intercept in the model by setting `intercept = TRUE/FALSE` as well as choose if the third stage of DECO algorithm ("refinement step") has to be performed (`refinement = TRUE/FALSE`). The number of threads used can be specified by setting `ncores` accordingly, while `m` indicates the number of submatrices X_i.

DECO_LASSO_MIX

```
  DECO_LASSO_MIX(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,r_2=0.001, ncores = ncores,
               refinement = TRUE, intercept=TRUE)
```

The input parameters are the same as the ones of `DECO_LASSO_R` function.

DECO_LASSO_C_PARALLEL

```
  DECO_LASSO_C_PARALLEL(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,r_2=0.001, ncores = ncores,
                        refinement = TRUE, intercept=TRUE, glmnet=TRUE, parallel_glmnet=FALSE)
```

Lasso coefficients can be computed using R function `glmnet` or a C++ implementation of the coordinate descent algorithm (still unstable) by setting `glmnet = TRUE/FALSE`. To use a parallelized version of `glmnet`, `parallel_glmnet` must be set to `TRUE` and it is not generally advised for small datasets, since run time

might be slower due to the communication between C++ and R. Coordinate descent algorithm always uses a parallelized version and its input parameter can be tuned by changing `precision` and `max_iter` parameters.

Note however that coordinate descent algorithm currently supports only matrices whit $n > p$. The m submatrices should then all have more rows than columns or an error is returned.

DECO_LASSO_R_CLUSTER

```
DECO_LASSO_R_CLUSTER(Y, X, p=p, n=n, lambda=lambda, r_1=r, r_2=0.001, clust=clust, ncores=ncores,
                     refinement = TRUE, intercept=TRUE)
```

The input parameters are the same as the ones of `DECO_LASSO_R` function. `clust` represents an object obtained, for instance, by `makePSOCKcluster` function.

Speed comparison

Since `DECO_LASSO_C_PARALLEL` function is entirely written in C++, it is faster than both `DECO_LASSO_MIX` and `DECO_LASSO_R`. Coordinate descent algorithm is still unstable, so `glmnet` parameter should be set to `TRUE`. If the dataset is small, setting `parallel_glmnet = FALSE` can lead to faster performances.

```
#Matrix 5x17; m=4; ncores=4
for (refinement in c(FALSE,TRUE)) {
  print(paste("Refinement is set to",refinement))
  print(benchmark(DECO_LASSO_R(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,
                               ncores = ncores, refinement = refinement),
            DECO_LASSO_MIX(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,
                           ncores = ncores, refinement = refinement),
            DECO_LASSO_C_PARALLEL(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,
                                  ncores = ncores, refinement = refinement),
            DECO_LASSO_C_PARALLEL(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,
                                  ncores = ncores, refinement = refinement, parallel_glmnet = TRUE),
            DECO_LASSO_C_PARALLEL(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,
                                  ncores = ncores, refinement = refinement, glmnet = FALSE),
            replications = 10, order="relative"))
}
```

```
## [1] "Refinement is set to FALSE"
##
## 1                                  DECO_LASSO_R(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
## 2                                DECO_LASSO_MIX(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
## 3                          DECO_LASSO_C_PARALLEL(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
## 4 DECO_LASSO_C_PARALLEL(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r, ncores = ncores, refiner
## 5         DECO_LASSO_C_PARALLEL(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r, ncores = ncores
##   replications elapsed relative user.self sys.self user.child sys.child
## 1           10   2.861       NA     2.446    0.353      0.167     0.370
## 2           10   2.349       NA     1.914    0.377      0.153     0.395
## 3           10   0.058       NA     0.102    0.003      0.000     0.000
## 4           10   0.115       NA     0.055    0.072      0.099     0.130
## 5           10   0.000       NA     0.002    0.001      0.000     0.000
## [1] "Refinement is set to TRUE"
```

```
##
## 5         DECO_LASSO_C_PARALLEL(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r, ncores = ncores
## 3                         DECO_LASSO_C_PARALLEL(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
## 4 DECO_LASSO_C_PARALLEL(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r, ncores = ncores, refiner
## 2                                 DECO_LASSO_MIX(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
## 1                                   DECO_LASSO_R(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
##   replications elapsed relative user.self sys.self user.child sys.child
## 5           10   0.016    1.000     0.056    0.000      0.000     0.000
## 3           10   0.065    4.062     0.107    0.002      0.000     0.000
## 4           10   0.115    7.187     0.071    0.054      0.102     0.119
## 2           10   2.347  146.687     1.911    0.372      0.161     0.407
## 1           10   3.086  192.875     2.633    0.387      0.150     0.443
```

On this example (with X being a 5×17 matrix), `DECO_LASSO_C_PARALLEL` with `glmnet = FALSE` is the fastest algorithm since the code is entirely in C++. Note that since the dataset is small, the not parallelized version of `DECO_LASSO_C_PARALLEL` when `glmnet = TRUE` is fastest. As predictable, `DECO_LASSO_R` and `DECO_LASSO_MIX` functions are significantly slower.

```
n<-1000
p<-10000
m<-8
X<-matrix(rnorm(n*p, sd=1), nrow=n)
eps<-rnorm(n, mean = 0, sd = sigma)
activeCoefs<-sample(c(0,1),p, prob=c(0.9, 0.1), replace=TRUE)  ##get coefficients that impact Y
coefs<-activeCoefs*1
Y<-X%*%coefs + eps
ncores <- 8
#Matrix 1000x10000; m=8; ncores=8
for (refinement in c(FALSE,TRUE)) {
  print(paste("Refinement is set to",refinement))
  print(benchmark(DECO_LASSO_R(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,
                               ncores = ncores, refinement = refinement),
            DECO_LASSO_MIX(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,
                           ncores = ncores, refinement = refinement),
            DECO_LASSO_C_PARALLEL(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,
                                  ncores = ncores, refinement = refinement),
            DECO_LASSO_C_PARALLEL(Y, X, p=p, n=n, m=m, lambda=lambda, r_1=r,
                                  ncores = ncores, refinement = refinement, parallel_glmnet = TRUE),
            replications = 5, order="relative"))
}
```

```
## [1] "Refinement is set to FALSE"
##
## 4 DECO_LASSO_C_PARALLEL(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r, ncores = ncores, refiner
## 2                                 DECO_LASSO_MIX(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
## 3                         DECO_LASSO_C_PARALLEL(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
## 1                                   DECO_LASSO_R(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
##   replications elapsed relative user.self sys.self user.child sys.child
## 4            5  36.900    1.000   151.093    1.045     34.143     0.847
## 2            5  48.146    1.305    13.316    3.294    238.425     6.116
## 3            5  48.589    1.317   166.002    0.321      0.000     0.000
## 1            5  58.581    1.588    24.263    3.000    241.973     5.683
## [1] "Refinement is set to TRUE"
##
```

```
## 4 DECO_LASSO_C_PARALLEL(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r, ncores = ncores, refiner
## 3                         DECO_LASSO_C_PARALLEL(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
## 2                                DECO_LASSO_MIX(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
## 1                                  DECO_LASSO_R(Y, X, p = p, n = n, m = m, lambda = lambda, r_1 = r,
##   replications elapsed relative user.self sys.self user.child sys.child
## 4            5  44.757    1.000   159.394    1.071     37.468     0.948
## 3            5  55.223    1.234   174.881    0.357      0.000     0.000
## 2            5  70.186    1.568    34.435    3.800    238.317     7.171
## 1            5  83.024    1.855    47.416    3.616    236.388     6.288
```

Note that the parallel version of `glmnet` is now faster.

Further development and improvement

- **Stable coordinate descent algorithm**: R function `glmnet` is exceptionally fast, but the communication between C++ and R slows down the computation. A stable C++ function to compute the Lasso regression coefficients would solve this problem.
- **Adaptive penalty**: following the original paper, a modification of BIC could be used to automatically tune the penalty λ.
- **Chunkwise access to the matrix**: `DECO_LASSO_R_CLUSTER` is an (initial) implementation of the algorithm to distribute the work load on several machines. This becomes necessary when the matrix X is too big to be stored in the RAM and it should then be accessed chunkwise.